IMPLEMENTASI BASISDATA DALAM *REAL-TIME SYSTEM*

Oleh : Adriansyah (13505070)

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15070@students.if.itb.ac.id

Abstrak

Basisdata merupakan salah satu elemen penting yang tidak bisa dipisahkan dari sebuah sistem. Basisdata dapat diibaratkan sebagai sebuah pondasi bagi sistem untuk menyediakan informasi dan data yang dibutuhkan, baik yang digunakan untuk menunjang sistem itu sendiri ataupun sebagai *output* untuk pengguna. Hingga saat ini, sangat banyak jenis aplikasi basisdata yang disediakan oleh berbagai *vendor software* terkemuka dunia. PostgreSQL, Oracle, dan MySQL adalah sebagian kecil dari berbagai aplikasi basisdata yang sering digunakan oleh berbagai perusahaan dan organisasi. Khusus untuk yang disebutkan terakhir, jenis aplikasi basisdata ini merupakan aplikasi yang *freeware*, yakni gratis dan tidak dipungut biaya dalam penggunaannya. Bergerak di bawah bendera GPL (*General Public License*), MySQL merupakan aplikasi basisdata yang paling banyak digunakan di seluruh dunia saat ini.

Keuntungan dari penggunaan basisdata dalam menunjang sebuah sistem sangatlah banyak. Walaupun basisdata cenderung bergerak “di belakang layar”, keberadaannya sangat menentukan kinerja sebuah sistem. Keefektifan dalam pengaksesan data, relasi dan hubungan antar data yang terstruktur rapi, penghematan tempat penyimpanan eksternal data merupakan salah satu alasan mengapa berbagai *vendor software* atau para *programmer* selalu berusaha untuk mengintegrasikan sistem atau *software* mereka dengan basisdata.

Dalam implementasinya di sebuah *real-time system*, model basisdata yang biasanya digunakan adalah *active database system* atau *real-time database system* yang masing-masing mempunyai ciri khas tersendiri. Adapun yang dimaksud dengan *real-time system* itu sendiri adalah sebuah sistem yang berjalan secara dinamik dan dalam waktu yang sebenarnya (sesuai dengan kenyataan). Sebuah *real-time system* juga dituntut untuk tidak hanya sekedar mampu menjalankan fungsinya saja namun juga harus bisa memperhatikan berbagai kondisi yang ada dan dapat merespon dalam rentang kebutuhan waktu tertentu sehingga dapat memuaskan keinginan pengguna. Salah satu contoh dari *real-time system* adalah sistem lampu lalu-lintas, sistem telekomunikasi, sistem transaksi antar bank, sistem komando militer, dan sebagainya.

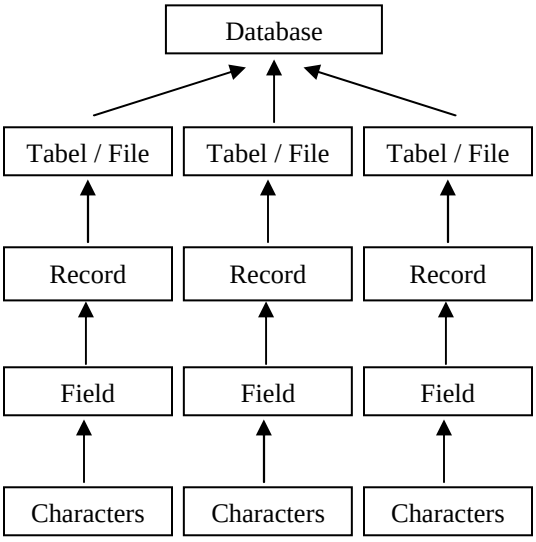
1. Pendahuluan

Dahulu jauh ketika belum dikenal konsep basisdata, berbagai jenis data dalam sistem diolah secara individu dan digunakan hanya ketika sistem tersebut membutuhkannya sehingga tidak dikenal adanya konsep penyimpanan data. Namun seiring dengan perkembangan teknologi di mana penyediaan data tidak hanya sebagai pelengkap tapi sudah menjadi sebuah *resource* yang sangat vital bagi performansi sebuah sistem, mulailah dikenalkan konsep basisdata.

Definisi basisdata adalah kumpulan dari berbagai jenis data yang saling berhubungan antara satu dengan yang lainnya dan diorganisasikan berdasarkan skema atau struktur tertentu.

Basisdata merupakan kumpulan tabel di mana setiap tabel diimplementasikan secara fisik sebagai sebuah *file*. Satu baris data pada tabel menyatakan sebuah *record* dan setiap atribut atau kolom menyatakan sebuah *field*. Sedangkan *characters* merupakan bagian dari data terkecil yang bisa berupa huruf, angka maupun karakter-karakter khusus yang membentuk sebuah *field*.

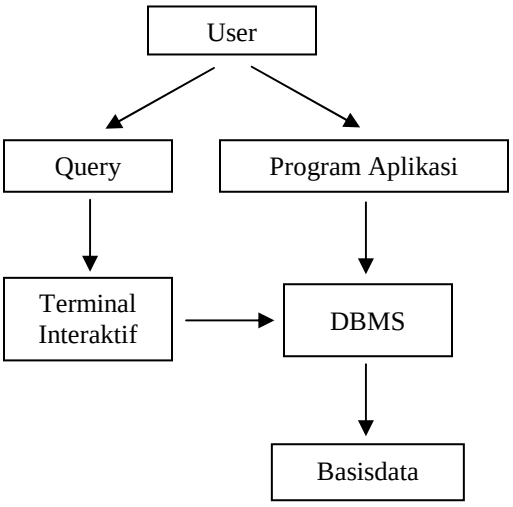
Diagram di bawah ini merupakan struktur dasar dari sebuah basisdata.



Untuk mengakses dan memanipulasi setiap data di dalam tabel, digunakan bahasa pemrograman SQL (*Structured Query Language*). Perintah SQL dibedakan menjadi 3 jenis, yaitu :

- DDL (*Data Definition Language*)
  Digunakan untuk mendefinisikan basisdata dan tabel. Dengan DDL, kita dapat membuat tabel, mengubah strukturnya, menghapus tabel, membuat indeks tabel, dan segala hal yang berkaitan dengan pembentukan struktur basisdata.
- DML (*Data Manipulation Language*)
  Digunakan untuk memanipulasi data, seperti menambahkan (*insert*), mengubah (*update*), menghapus (*delete*) atau mengambil dan mencari data (*select*).
- DCL (*Data Control Language*)
  Digunakan untuk mendefinisikan daftar pengguna yang boleh mengakses basisdata serta apa saja hak dan kewenangannya.

Pengguna dapat berkomunikasi dengan basisdata dengan 2 cara, yaitu melalui *query* (bahasa SQL) dan melalui program aplikasi. Diagram di bawah ini merupakan skema hubungan antara pengguna dengan basisdata.



Manfaat yang dapat dirasakan dengan adanya basisdata pada sebuah sistem adalah :

- Manajemen dan pemeliharaan data yang baik karena seluruh data disatukan sebagai sebuah *resource*.
- Kontrol akses dan manipulasi terhadap data lebih mudah karena seluruh data tersebut terpusat.
- Mengurangi duplikasi data (*data redundancy*).

- Relasi dan hubungan antar data dapat dikembangkan lebih kompleks.
- Menghemat tempat penyimpanan eksternal data.

Adapun tipe *file* yang dikenal dalam sebuah basis data terbagi menjadi 5 macam, yaitu :

1. *File* induk (*master file*)
   a. file induk acuan (*reference master file*), yaitu file induk yang *record*-nya relatif jarang berubah nilainya (statis). Contohnya adalah *file* matakuliah program studi informatika dan *file* merk mobil Eropa.
   b. file induk dinamik (*dynamic master file*), yaitu file induk yang *record*-nya sering berubah nilainya. Contohnya adalah *file* inventaris barang masuk dan keluar pabrik.
2. *File* transaksi (*transaction file/input file*), yaitu *file* yang digunakan untuk mencatat dan merekam data dari hasil transaksi yang terjadi.
3. *File* laporan (*report file/output file*), yaitu *file* yang berisi data dari informasi yang akan ditampilkan.
4. *File* sejarah (*history file/archival file*), yaitu *file* yang berisi data di masa lalu tapi masih disimpan sebagai arsip.
5. *File* pelindung (*backup file*), yaitu *file* yang berisi salinan atau kopian dari data yang masih aktif di dalam basisdata. Tujuannya adalah sebagai cadangan jika sewaktu-waktu data di dalam basisdata hilang atau mengalami kerusakan.

2. *Real-Time System*

Real-time system dapat didefinisikan sebagai sebuah sistem yang tidak hanya berorientasi terhadap hasil (*output*) yang dikeluarkan tapi di sana juga sistem dituntut untuk dapat bekerja dengan baik dalam kebutuhan waktu tertentu.

Di dalam *real-time system*, waktu merupakan faktor yang sangat penting untuk diperhatikan. Faktor waktu menjadi sesuatu yang sangat kritis dan sebagai tolak ukur baik-tidaknya kinerja keseluruhan sistem tersebut. Akan tetapi, ada satu hal yang perlu diingat, *real-time system* tidak sama dengan *fast-system*. *Fast-system* adalah sistem yang bekerja dalam waktu yang sesingkat-singkatnya yang dalam artian semakin cepat *output* yang dihasilkan oleh sistem tersebut berarti semakin baik kinerjanya.

Berbeda dengan *fast-system*, *real-time system* bekerja dalam periode dan waktu *deadline* tertentu sehingga belum tentu semakin cepat *output* yang dihasilkan berarti menunjukkan sistem tersebut bekerja dengan baik. Adapun contoh dari *real-time system* adalah sistem perbankan, sistem pengontrol pesawat udara, sistem otomasi pabrik, dan sebagainya.

2.1. Model *Real-time system*

Model *real-time system* dapat dibagi menjadi 3 jenis berdasarkan prioritas ketika menemui waktu *deadline*, yaitu *hard real-time system*, *soft real-time system*, dan *firm real-time system*.

Dalam *hard real-time system*, kebenaran eksekusi program dan waktu *deadline* (*hard deadline*) menjadi sangat kritis dan menentukan performansi seluruh sistem. Jika sistem tersebut tidak mampu memenuhi waktu *deadline* yang telah ditentukan, maka akan berakibat fatal terhadap seluruh sistem. Kondisi ini dikenal dengan istilah *catastropic consequences* (bencana besar). Oleh karena itu, desain dan reliabilitas dari sebuah *hard real-time system* harus benar-benar diperhatikan dan dievaluasi dengan baik.

Berbeda dengan *hard real-time system*, *soft real-time system* jauh lebih toleran dan tidak terlalu kritis ketika sistem tidak mampu memenuhi waktu *deadline* (*soft deadline*). Sistem tersebut tidak akan *failure* walaupun waktu *deadline*-nya tidak terpenuhi. Selain itu, sistem ini tetap akan bekerja dan menyelesaikan tugasnya meskipun waktu *deadline*-nya sudah lewat.

Lain halnya dengan *firm real-time system*. Walaupun sama dengan *soft real-time system* dalam hal toleransi waktu *deadline*, tapi sistem model ini tidak akan bekerja dan menyelesaikan tugasnya ketika waktu *deadline* sudah lewat. Dengan kata lain, sistem ini akan berhenti bekerja tapi tidak akan menyebabkan *failure* pada keseluruhan sistem.

Pada *real-time system* dikenal istilah *tardy tasks* dan *miss-percentage*. *Tardy tasks* adalah tugas yang tidak dapat dikerjakan dan dieksekusi oleh sistem dalam waktu *deadline* tertentu. Sedangkan yang dimaksud dengan *miss-percentage* adalah persentase dari *tardy tasks* terhadap seluruh tugas yang harus dikerjakan oleh sistem.

Pada *soft real-time system*, *miss-precentage* bertambah secara eksponensial seiring dengan jumlah tugas yang harus dikerjakan. Sedangkan pada *firm real-time system* di mana *tardy task* tidak dikerjakan oleh sistem, *miss-percentage* bertambah secara polinomial. Hal ini menunjukkan bahwa tingkat *miss-percentage* pada *soft real-time system* lebih tinggi daripada *firm real-time system*.

Oleh karena tuntutan atas waktu yang tinggi dari sebuah *real-time system*, maka dibutuhkan model basisdata yang berbeda dengan basisdata konvensional. Pada *real-time system* dikenal model basisdata *active database system* dan *real-time database system* yang dalam penggunaannya bisa berdiri sendiri ataupun bisa dikombinasikan antar keduanya.

3. *Active database system*

Active database system adalah sistem basisdata yang mampu menghasilkan sebuah *action* tertentu berdasarkan kondisi yang terjadi.

Berbeda dengan basisdata konvensional yang tergantung secara manual kepada program aplikasi atau pengguna untuk mengeksekusi perintah *query*, *active database system* didesain untuk mampu melakukan eksekusi secara otomatis berdasarkan kondisi tertentu dan sama sekali tidak tergantung kepada program aplikasi atau pengguna dalam mengeksekusi perintah.

Dalam *active database system* dikenal 3 istilah yang cukup penting, yaitu *event*, *condition*, dan *action*, atau yang lebih sering dikenal dengan istilah ECA (*Event-Condition-Action*) *Rule*.

3.1. ECA *Rule*

ECA *Rule* adalah sebuah aturan di mana *active database system* akan mendeteksi sebuah kejadian terlebih dahulu, kemudian mengecek pemilihan kondisi dari kejadian tersebut. Jika kondisinya terpenuhi, maka *active database system* akan melakukan *action* tertentu. Di bawah ini merupakan algoritma sederhana dari ECA *Rule*.

on *<event A>*
if *<condition B>*
then *<action C>*

3.1.1. *Event*

Event adalah kejadian yang berlangsung pada suatu waktu tertentu. *Event* dibagi menjadi dua jenis, yaitu :

1. *Primitive Events*, yaitu kejadian yang telah ditentukan. Event ini dibagi lagi menjadi 3 macam, yaitu :
   a. *Database Events*
      Kejadian yang berhubungan dengan operasi-operasi manipulasi pada basisdata (*insertion*, *update*, *deletion*, *retrieval*).
   b. *Temporal Events*
      Kejadian yang dipicu oleh syarat-syarat tertentu. Contohnya ialah kejadian B terjadi 5 menit setelah kejadian A, kejadian C dilakukan setiap jam 10 pagi, dan sebagainya.
   c. *Explicit Events*
      Kejadian yang terjadi di luar basisdata (eksternal). Contohnya ialah pembacaan data barang belanja oleh sensor sewaktu pembayaran di kasir.

2. *Composite Events*
   Kumpulan dari *primitive events* yang dihubungkan dengan operator logika seperti *and* atau *or*.

3.1.2. *Condition*

Sebelum sebuah *action* dieksekusi oleh *active database system*, kondisi dari suatu kejadian harus dievaluasi dulu. Jika pada evaluasi tersebut menghasilkan nilai *true*, maka *action* akan dieksekusi. Dan sebaliknya, jika evaluasi menghasilkan nilai *false*, maka *action* tersebut tidak akan dieksekusi.

Pengevaluasian kondisi hanya bisa dilakukan setelah adanya pendeteksian terhadap suatu kejadian atau jika seluruh kejadian pada sistem telah selesai.

3.1.3. *Action*

Action dilakukan setelah kondisi terpenuhi. *Action* dapat berupa sebuah *query* (i*nsertion*, *update*, *deletion*, *retrieval*) atau berupa pemanggilan ke fungsi atau prosedur yang lain.

3.2. Mode Eksekusi

Pada *active database system* dikenal beberapa mode eksekusi. Beberapa di antaranya ialah

coupling mode, *cascade triggering*, dan *priority mode*.

3.2.1. *Coupling Mode*

Dalam ECA *Rule*, *event* selalu terjadi lebih dulu sebelum *condition* dan *action* (proses sekuensial). Dengan *coupling mode*, *condition* dan *action* bisa ditempatkan relatif tergantung pada kejadian transaksi pemicu (*triggering transaction event*). *Coupling mode* terbagi menjadi 3 jenis, yaitu :

- *Immediate*
  Transaksi akan ditunda sampai *condition* selesai dievaluasi dan *action* selesai dieksekusi. Selain itu, *action* juga tidak tergantung kepada hasil dari evaluasi kondisi.
- *Deferred*
  Condition dan *action* dijalankan ketika *triggering transaction* berakhir.
- *Detached*
  Condition dan *action* dijalankan di dalam satu atau transaksi yang berbeda.

3.2.2. *Cascade Triggering*

Casacade triggering adalah situasi ketika sebuah *event* memicu sebuah *action* yang *action* tersebut menjalankan lagi sebuah *event* lain yang memicu *action* yang berbeda. Situasi seperti ini dapat menyebabkan lingkaran *event* yang tidak akan pernah berhenti (*infinite circular triggering*).

Salah satu cara untuk mencegah terjadinya *cascade triggering* adalah dengan menentukan limit atau batas kedalaman dari *triggering event* sehingga pada suatu saat *triggering event* tersebut akan berhenti.

3.2.3. *Priority Mode*

Di dalam *active database system*, bisa saja ditemukan beberapa *event* yang mengalami konflik atau bentrok. Untuk mengatasi masalah ini, *active database system* harus menentukan sebuah aturan prioritas yang mengatur urutan proses dari *event-event* tersebut.

Aturan prioritas ini bisa ditentukan sendiri oleh pengguna atau ditentukan secara *default* oleh *active database system*. Salah satu contoh *priority mode* berdasarkan waktu adalah *event* yang terjadi lebih dulu mempunyai tingkat prioritas yang lebih tinggi.

4. *Real-Time Database System*

Real-time database system memiliki karakteristik yang cukup berbeda dengan *active database system*. Dalam *real-time database system*, dikenal berbagai proses seperti *scheduling*, *scheduler*, *admission control*, manajemen *buffer*, dan sebagainya. Proses-proses inilah yang membedakan *real-time database system* dengan *active database system*.

4.1. Transaksi dan *Scheduling*

Transaksi adalah kumpulan dari berbagai *action* yang bertujuan untuk melakukan operasi *read* dan *write* pada basisdata. Setiap transaksi mentransformasikan setiap *consistent-state* ke *consistent-state* yang lain.

Dalam transaksi, dikenal istilah ACID (*Atomicity, Consistency, Isolation, and Durability*). ACID adalah properti yang terkait dengan transaksi. Adapun definisi dari masing-masing istilah tersebut adalah :

- *Atomicity* : Semua operasi dalam transaksi diperlakukan sebagai sebuah individu.
- *Consistency* : Sebuah transaksi mampu memelihara keintegrasian dalam basisdata.
- *Isolation* : Sebuah transaksi mampu mengeksekusi beberapa *action* secara bersamaan tapi tidak menimbulkan konflik antar *action* tersebut.
- *Durability* : Semua perubahan yang dihasilkan oleh transaksi mampu disimpan dalam basisdata.

Sebuah *action* yang sekuensial dan berada di tiap transaksi yang berbeda dikenal dengan istilah *schedule*. *Schedule* dalam *real-time database system* diatur oleh *scheduler*. *Scheduler* adalah sebuah algoritma untuk mengatur dan menentukan berbagai aktivitas yang terjadi dalam basisdata, seperti waktu eksekusi, deadline, urutan eksekusi, dan sebagainya. Sebuah *scheduler* dapat dikatakan bekerja secara optimal apabila ia mampu untuk mengerjakan seluruh tugas yang telah ditentukan.

Schedule bisa ditentukan dengan dua cara, yaitu secara statik dan dinamik. *Schedule* dikatakan statik jika sebelumnya *schedule* tersebut telah ditentukan oleh pengguna pada saat *off-line*. Penentuan *schedule* secara statik juga membutuhkan pengetahuan tentang seluruh tugas

yang akan dikerjakan oleh sistem tersebut sampai selesai.

Metode statik sangat cocok jika dalam sistem tersebut tidak ada perubahan mengenai tugas dan prioritas yang akan dikerjakan oleh sistem pada saat *run-time*. Salah satu cara yang digunakan metode ini ialah dengan menyimpan seluruh tugas yang akan dikerjakan pada saat *run-time* dalam satu tabel sehingga sistem cukup merujuk dari tabel tersebut mengenai prioritas tugas yang akan dikerjakan.

Lain halnya dengan metode statik, penentuan *schedule* secara dinamik dilakukan pada saat *run-time*. Dengan metode ini, sistem dapat bekerja dengan lebih fleksibel dan mampu menangani berbagai kejadian yang tidak bisa diprediksi sebelumnya. Selain itu, sistem juga mempunyai kemampuan *error-recovery* dan dapat bekerja dalam periode waktu yang tidak pasti.

4.2. *Deadlock*

Salah satu permasalahan yang sering timbul dalam basisdata ialah *deadlock*. *Deadlock* adalah suatu keadaan di mana beberapa transaksi saling menunggu satu sama lainnya sehingga membentuk sebuah siklus yang mengakibatkan sistem menjadi *hang*. Contoh kasusnya adalah sebagai berikut : Transaksi Ti akan membaca data A yang sedang ditulis oleh transaksi Tj sedangkan Tj akan akan membaca data B yang sedang ditulis oleh Ti. Pada kondisi ini, tidak ada transaksi yang bekerja karena semuanya saling menunggu.

Untuk mengatasi *deadlock*, ada beberapa cara yang bisa dilakukan. Di antaranya ialah :

1. Membatalkan *tardy transaction* yang pertama kali terdeteksi dalam siklus *deadlock* tersebut. Jika tidak ada *tardy transaction* dalam siklus *deadlock*, batalkan transaksi dengan waktu *deadline* paling lama atau transaksi dengan waktu *deadline* paling sebentar atau transaksi dengan tingkat kritis paling rendah.
2. Membatalkan transaksi yang paling tidak mungkin untuk dikerjakan. Jika semua transaksi memungkinkan untuk dikerjakan, batalkan transaksi dengan tingkat kritis paling rendah.

4.3. Resolusi Konflik

Dalam setiap *event* selalu ada kemungkinan timbulnya konflik antar transaksi. Biasanya konflik yang sering terjadi adalah beberapa transaksi bekerja berbarengan dengan operasi yang berbeda (*read/write*). Pada sumber daya basisdata yang dipakai bersama oleh beberapa aplikasi (*shared database resource*), konflik ini biasanya akan menyebabkan *roll-back* pada sistem.

Salah satu solusi yang bisa dipakai untuk menangani konflik ini adalah dengan menggunakan teknik *Wound-Wait* yang ditemukan oleh Rosenkrantz. Teknik ini biasa digunakan untuk menghindari *deadlock*.

Dalam perkembangannya, teknik ini kemudian dimodifikasi oleh Abbott dan Garcia-Molina, yang sekarang lebih dikenal dengan nama *High-Priority* (HP) dan *Priority-Abort* (PA). Adapun algoritma dari HP dan PA seperti ditunjukkan pada algoritma di bawah ini.

Misalkan :
P(Ti) : prioritas transaksi Ti.
Tr meminta kunci dari data D

if (tidak ada konflik) ***then***
    Tr mengakses D
else
    Th memegang data yang diminta lalu menyelesaikan konfliknya sesuai dengan algoritma di bawah ini :

    if (P(Tr) > P(Th)) ***then***
        Th dibatalkan
    else
        Tr menunggu kuncinya

Pada algoritma di atas, prioritas yang lebih rendah (Th) dibatalkan dengan tujuan untuk memberikan *resource* miliknya kepada prioritas yang lebih tinggi (Tr). Namun prioritas yang lebih rendah diperbolehkan untuk menunggu prioritas yang lebih tinggi.

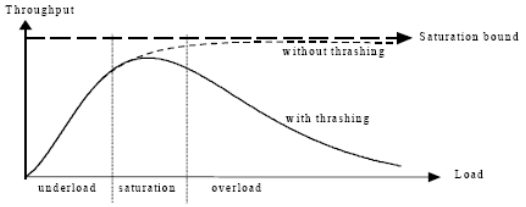
4.4. *Admission Control*

Masalah lain yang biasanya muncul dalam *real-time database system* ialah jumlah transaksi yang melebihi kapasitas sumber daya sistem (CPU, *buffer*, I/O *queues*, dan basisdata) sehingga menyebabkan *overload*.

Overload dapat menyebabkan *thrashing* pada sistem. *Thrashing* adalah suatu keadaan di mana dengan meningkatnya jumlah transaksi dapat menyebabkan menurunnya *throughput* dan/atau menurunnya jumlah transaksi yang dapat memenuhi waktu *deadline*. *Thrashing* dapat disebabkan oleh :

- Adanya konflik antar data
- Adanya konflik antar sumber daya fisik pada sistem (*physical resources*) meskipun tidak terjadi konflik antar data. Hal ini dikarenakan banyaknya transaksi yang menggunakan sumber daya pada sistem.
- Meningkatnya jumlah transaksi.

Grafik di bawah ini menunjukkan pengaruh *overload* terhadap *throughput* (fungsi *load-throughput*).



Penjelasan grafik :

- Pada kondisi *underload*, *throughput* naik hampir secara linear.
- Ketika sumber daya sistem sudah digunakan semuanya, sistem mencapai batas jenuh (*saturation bound*) dan fungsi *throughput* mulai bergerak tetap (statis).
- Pada kondisi *overload*, *throughput* menurun dan sistem mengalami *thrashing effect*.

Ada beberapa teknik yang bisa digunakan untuk mencegah *overload* dan *thrashing*. Beberapa di antaranya adalah :

- *Fixer upper bound*
  Menentukan jumlah maksimal transaksi yang masuk ke dalam sistem.
- *Feedback control*
  Memonitor secara dinamik tingkat konkurensi dan kegiatan sistem. Heiss dan Wagner menemukan dua algoritma yang bisa diterapkan untuk teknik ini, yaitu *Incremental Steps* (IS) dan *Parabola Approximation* (PA).
- *Adaptive-Earliest-Deadline* (AED)
  Menggunakan mekanisme *feedback* untuk menstabilkan kondisi *overload*. Teknik ini kemudian dimodifikasi oleh Haritsa menjadi *Hierarchical-Earliest-Deadline* (HED) yang menggabungkan nilai transaksi dengan *deadline*-nya.

4.5. Manajemen Memori

Manajemen memori harus sangat diperhatikan dalam sebuah *real-time system* karena sangat menentukan kinerja sistem secara keseluruhan. Manajemen memori dapat ditinjau dari tiga aspek, yaitu hak transaksi, alokasi *buffer*, dan pergantian *buffer*. Pada kali ini, penulis hanya akan membahas mengenai alokasi *buffer* dan pergantian *buffer*.

Alokasi *buffer* bertujuan untuk mendistribusikan *buffer* yang tersedia untuk berbagai transaksi yang terjadi secara bersamaan. Sedangkan pergantian *buffer* bertujuan untuk meminimalisasi tingkat kesalahan *buffer*.

Dalam *real-time system*, tujuan dari *buffer* adalah untuk mempersingkat waktu transaksi dan untuk meningkatkan jumlah persentase transaksi dalam memenuhi kebutuhan waktu yang telah ditentukan. Selain itu, *buffer* dapat menjadi salah satu alternatif penyimpanan data jika kita hanya memiliki kapasitas yang terbatas di memori utama. Oleh karena itu, kita harus benar-benar memperhatikan kerja dari manajemen *buffer* agar dapat memenuhi berbagai tujuan di atas.

Adapun proses kerja *buffer* itu sendiri ialah ketika transaksi baru datang, *buffer manager* bertanggung jawab untuk menyediakan beberapa blok *buffer* untuk dialokasikan ke transaksi yang baru datang tersebut. Jika tidak tersedia blok *buffer* yang kosong, *buffer manager* harus menentukan data mana di dalam *buffer* yang akan ditimpa oleh data yang baru.

Namun masalah akan muncul jika pada transaksi yang masuk diberikan prioritas sehingga setiap transaksi tidak bisa diperlakukan sama. Hal ini mengakibatkan *buffer manager* harus lebih hati-hati dalam menentukan data mana dalam *buffer* yang akan ditimpa oleh data yang baru.

Ada dua teknik yang bisa digunakan dalam manajemen memori, yaitu *Global Least Recently Used* (G-LRU) *Buffer Management* dan *Priority Memory Management* (PMM).

4.5.1. *Global Least Recently Used* (G-LRU) *Buffer Management*

Dalam G-LRU *Buffer* Management, ketika transaksi datang dan tidak ada blok *buffer* yang tersedia, maka blok *buffer* yang berisi data paling kecil dan baru datang (*least recently used*) yang akan ditimpa.

Algoritma G-LRU *Buffer* Management sangat simpel dan biasanya digunakan untuk sistem basisdata komersial.

Dalam teknik ini, prioritas transaksi diatur sedemikian rupa (secara dinamik) sehingga setiap blok *buffer* mempunyai level prioritasnya masing-masing. Ketika sistem pertama kali dijalankan, semua blok *buffer* kosong dan disusun sebagai sebuah *list* yang kosong.

Ketika transaksi baru datang dengan prioritas P dan akan dialokasikan ke *buffer*, transaksi tersebut ditempatkan di antrian *least recenty used* yang mempunyai prioritas P. Agar setiap blok *buffer* tidak mempunyai antrian prioritas yang terlalu banyak, maka setiap antrian mempunyai rentang prioritas tertentu.

Kemudian selanjutnya pada saat *buffer manager* ingin menimpa data yang lama dengan data yang baru, maka *buffer manager* akan mulai mencari di antrian *least recently used* dengan prioritas yang paling rendah. Jika data yang dicari pada *buffer* lebih dulu waktunya daripada data yang baru, maka *buffer manager* akan melanjutkan pencarian ke prioritas yang lebih tinggi. Jika tidak lebih dulu waktunya daripada data yang baru, maka data yang lama tersebut akan ditimpa.

Jika benar-benar tidak tersedia lagi blok *buffer*, maka *buffer manager* akan menunda atau membatalkan transaksi dengan prioritas yang paling rendah.

4.5.2. *Priority Memory Management* (PMM)

Dalam PMM, ada dua teknik yang bisa digunakan, yaitu teknik *Max* dan *Min-Max* yang keduanya berjalan di bawah aturan EDF.

- *Max*
  Dengan teknik ini, sebuah transaksi harus dialokasikan memori sebesar kebutuhan maksimalnya selama proses eksekusi atau tidak dialokasi memori sama sekali.
- *Min-Max*
  Dengan teknik ini, transaksi dengan prioritas rendah boleh dieksekusi dengan alokasi memori sebesar kebutuhan minimumnya. Dan transaksi dengan prioritas tinggi dialokasikan memori sebesar kebutuhan maksimalnya. Keuntungan teknik ini adalah sistem dapat menjalankan lebih banyak transaksi.

Pemilihan kedua teknik di atas tergantung pada beban kerja sistem pada saat itu. Pada teknik *Min-Max*, proses dimulai dari transaksi yang paling tinggi prioritasnya. Jika ada *buffer* yang tersisa dari proses eksekusi transaksi tadi, maka selanjutnya akan dialokasikan ke transaksi berikutnya, dimulai dari transaksi yang paling tinggi prioritasnya.

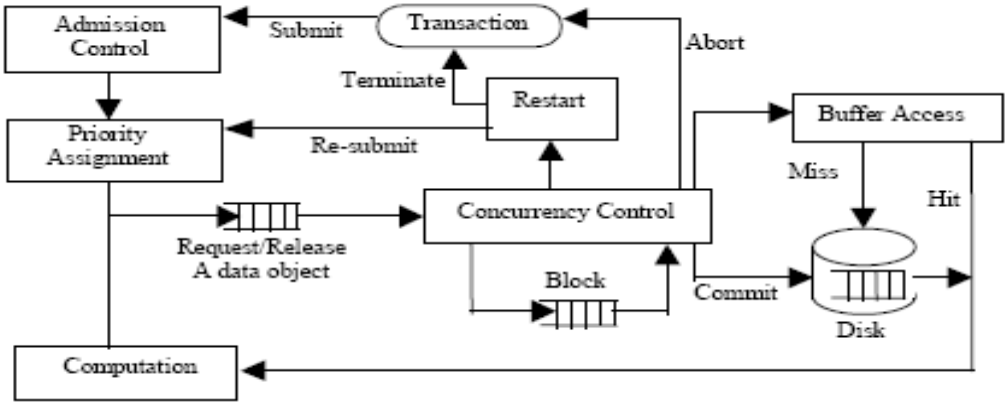
Proses alokasi memori ini akan berakhir jika semua memori sudah dialokasikan atau jika semua transaksi sudah menerima kebutuhan maksimal memori masing-masing.

Di akhir proses alokasi memori dengan teknik *Min-Max*, semua transaksi dengan prioritas lebih tinggi akan mendapatkan kebutuhan maksimal memorinya sedangkan transaksi dengan prioritas lebih rendah hanya akan mendapatkan kebutuhan minimum memorinya.

Lain halnya dengan teknik *Max*. Pada teknik ini manajemen memori lebih mudah karena dapat mengabaikan transaksi-transaksi dengan prioritas rendah. Akan tetapi, teknik ini akan menyulitkan di tingkat *multiprogramming* jika setiap transaksi membutuhkan memori yang besar. Oleh karena itu, teknik ini lebih disukai jika memori yang dimiliki oleh sistem sangat berlimpah.

Kita bisa menggabungkan dua teknik di atas dengan cara men-*switch* sesuai dengan keadaan pada saat itu. Sebagai *default*, kita dapat memilih teknik *Max*. Kemudian kita akan mengganti ke teknik *Min-Max* jika semua kondisi di bawah ini terpenuhi, yaitu :

- Satu atau lebih transaksi tidak bisa memenuhi waktu *deadline*.



Gambar 1. Model Real-time Database System

- Sumber daya pada sistem menjadi sangat rendah.
- Ada memori yang konflik seperti yang telah dijelaskan sebelumnya pada subbab *Admission Control*.
- Transaksi telah selesai jauh sebelum waktu *deadline*.

4.6. Model *Real-Time Database System*

Gambar 1 di atas menunjukkan model *real-time database system*. Sebagaimana yang telah diperlihatkan pada model tersebut, setiap transaksi yang masuk ke dalam sistem harus melewati beberapa tahapan, yaitu :

- Setiap transaksi baru harus melewati mekanisme *Admission Control* yang akan memonitor dan mengatur jumlah transaksi yang aktif dan berjalan bersamaan dalam sistem untuk mencegah timbulnya *thrashing*.
- Penentuan tingkat prioritas untuk setiap transaksi baru atau transaksi yang diulang kembali.
- Sebelum transaksi dijalankan, transaksi tersebut harus masuk ke *concurrency control* untuk disinkronisasi. Jika permintaan memori dari transaksi tidak bisa dipenuhi, maka transaksi tersebut akan ditempatkan di antrian. Transaksi akan diaktifkan kembali jika blok memori sudah tersedia.
- Jika transaksi meminta data yang tidak ada di memori utama, maka transaksi tersebut akan ditempatkan di antrian. Transaksi akan diaktifkan kembali jika data yang diminta sudah tersedia dan jika tidak ada lagi transaksi yang lebih tinggi prioritasnya yang aktif.
- Ketika operasi transaksi sudah selesai dijalankan, transaksi tersebut akan menyimpan hasilnya dan membebaskan semua data miliknya.
- Transaksi boleh dibatalkan atau diulang kembali beberapa kali. Ada beberapa jenis pembatalan transaksi, yaitu :
  - *Terminating abort*
    - Pembatalan karena melewati waktu *deadline*.
    - Pembatalan diri sendiri (*self abort*) karena kondisi tertentu.
  - *Non-terminating abort*
    Pembatalan karena *deadlock* atau ada konflik data. Pada kondisi ini, transaksi akan diulang kembali jika ada kemungkinan transaksi tersebut bisa memenuhi waktu *deadline*.

5. Masalah dan Solusinya

Masalah yang timbul ketika mengimplementasikan basisdata ke dalam *real-time system* sangatlah beragam. Mulai dari masalah waktu *deadline*, manajemen memori, prioritas transaksi, kontrol terhadap transaksi yang konflik, dan sebagainya.

Berikut ini merupakan rincian sebagian besar masalah yang biasanya muncul pada saat mengintegrasikan *active database system* dan/atau *real-time database system* ke dalam sistem.

- *Deadlock*
- *Cascade triggering*
- *Overload*
- Konflik antar transaksi
- *Thrashing*
- Konflik antar sumber daya pada sistem

- Penyediaan memori (*buffer*) yang tidak memadai
- Transaksi tidak bisa memenuhi waktu *deadline*

Walaupun sebagian besar masalah tersebut sudah dipaparkan solusinya di masing-masing bab/subbab, penulis mencoba untuk menyimpulkan kembali solusi dari berbagai masalah tersebut. Di bawah ini merupakan solusi atas permasalahan yang telah penulis sebutkan di atas.

- Menentukan limit dari *triggering event* untuk menghindari *cascade triggering*.
- Menggunakan teknik *fixer upper bound*, *feedback control* atau *adaptive-earliest-deadline* untuk mencegah *overload* dan *thrashing*.
- Menggunakan teknik *Global Least Recently Used* (G-LRU) *Buffer Management* atau *Priority Memory Management* (PMM) untuk masalah penyediaan memori.
- Menggunakan teknik *High-Priority* (HP) dan *Priority-Abort* (PA) untuk menangani konflik antar transaksi.

6. Kesimpulan

Untuk mengimplementasikan basisdata ke dalam *real-time system* ternyata tidaklah semudah yang dibayangkan. Banyak sekali faktor-faktor yang harus diperhatikan, baik itu faktor internal maupun faktor eksternal sistem.

Oleh karena itu, kita harus benar-benar cermat dalam memilih model basisdata apa yang paling tepat untuk diimplementasikan ke dalam sistem tersebut agar ke depannya sistem yang telah dirancang dapat menjadi sebuah sistem yang kokoh.

DAFTAR PUSTAKA

1. Aldarmi, Saud A. 1998. *Real-Time Database Systems : Concepts and Design*. *http://www.homepage.cs.uri.edu/courses/attic/csc436/aldarmi98realtime.pdf*
   Tanggal akses : 30 Desember 2006
   Waktu akses : 15:00

2. Irmansyah, Faried. 2003. *Pengantar Database*. *http://ilmukomputer.com/umum/faried-database.php*
   Tanggal akses : 29 Desember 2006
   Waktu akses : 10:00

3. Munir, Rinaldi. 2006. *Diktat Kuliah IF2153 Matematika Diskrit*. Bandung.

4. Ridha, Muhammad. 2005. *Active Database for Real-Time Computing*. *http://www.ilmukomputer.com/umum/ridha-activedb.php*
   Tanggal akses : 29 Desember 2006
   Waktu akses : 10:00

5. Sidik, Betha. 2005. *MySQL Untuk Pengguna, Administrator, dan Pengembang Aplikasi Web*. Bandung : Informatika.